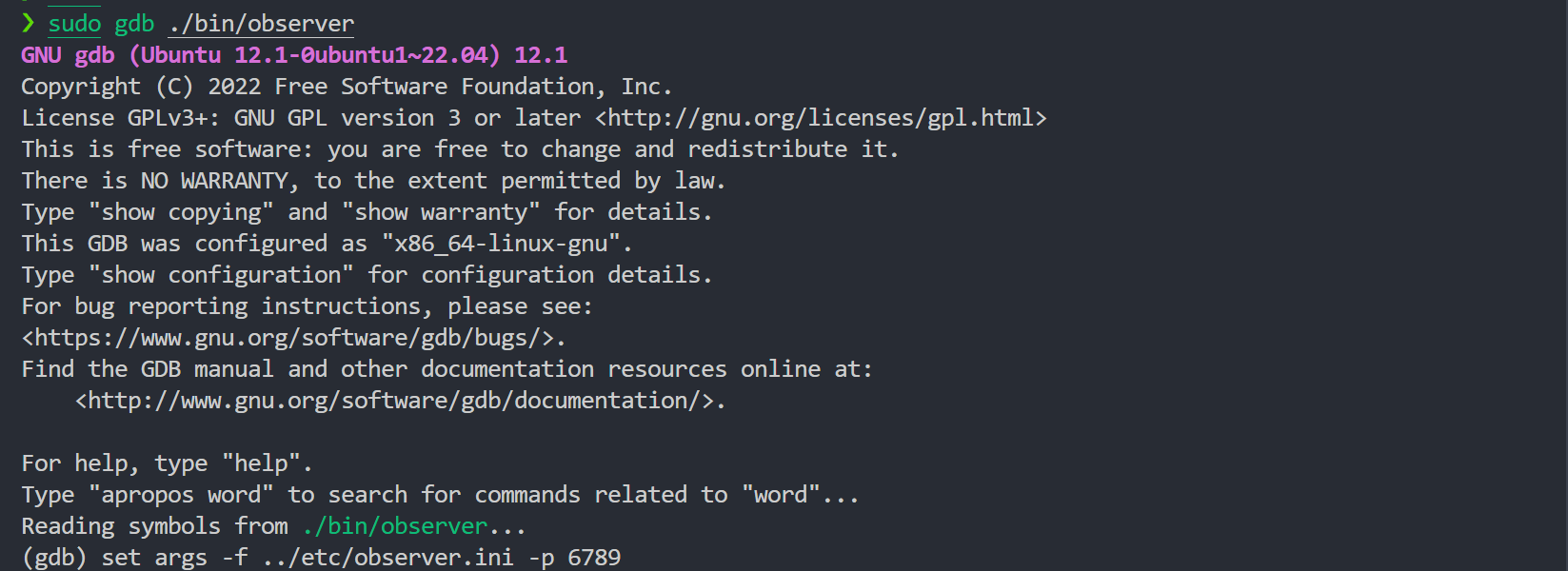
1. 分析MiniOB的Create Table实现原理，要画出Create Table实现的过程，并进行相应的描述。

启动Gdb调试 gdb ./bin/observer (或者用vscode 进行调试)

设置启动参数：set args -f ../etc/observer.ini -p 6789



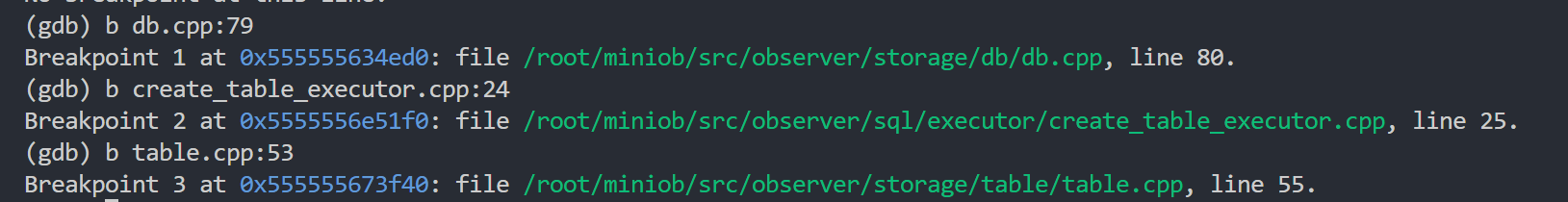
根据一个sql查询执行的过程，通过前面的语法解析，语法树生成，执行计划生成等最终会有相应的执行器(executor)去执行该sql。

在创建表的执行器中关键函数execute处设置断点 (语法 ： b 函数名/文件名:行号)

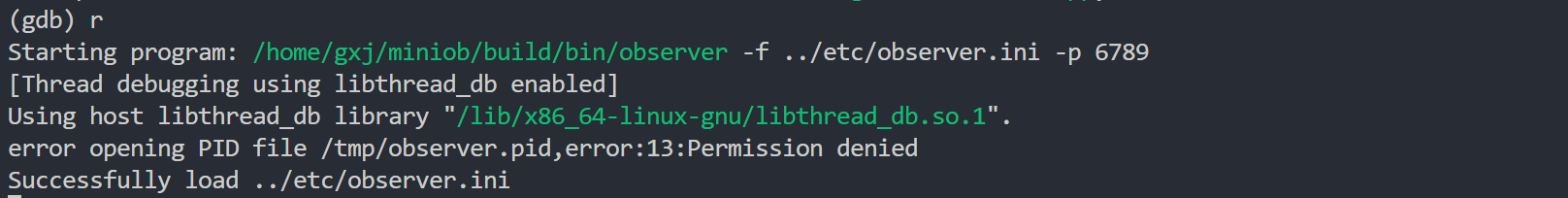
b create\_table\_executor.cpp:24 (或者b CreateTableExecutor::execute)

b db.cpp:79 (或者b Db::create\_table)

b table.cpp:53 (或者 b Table::create)



启动服务端



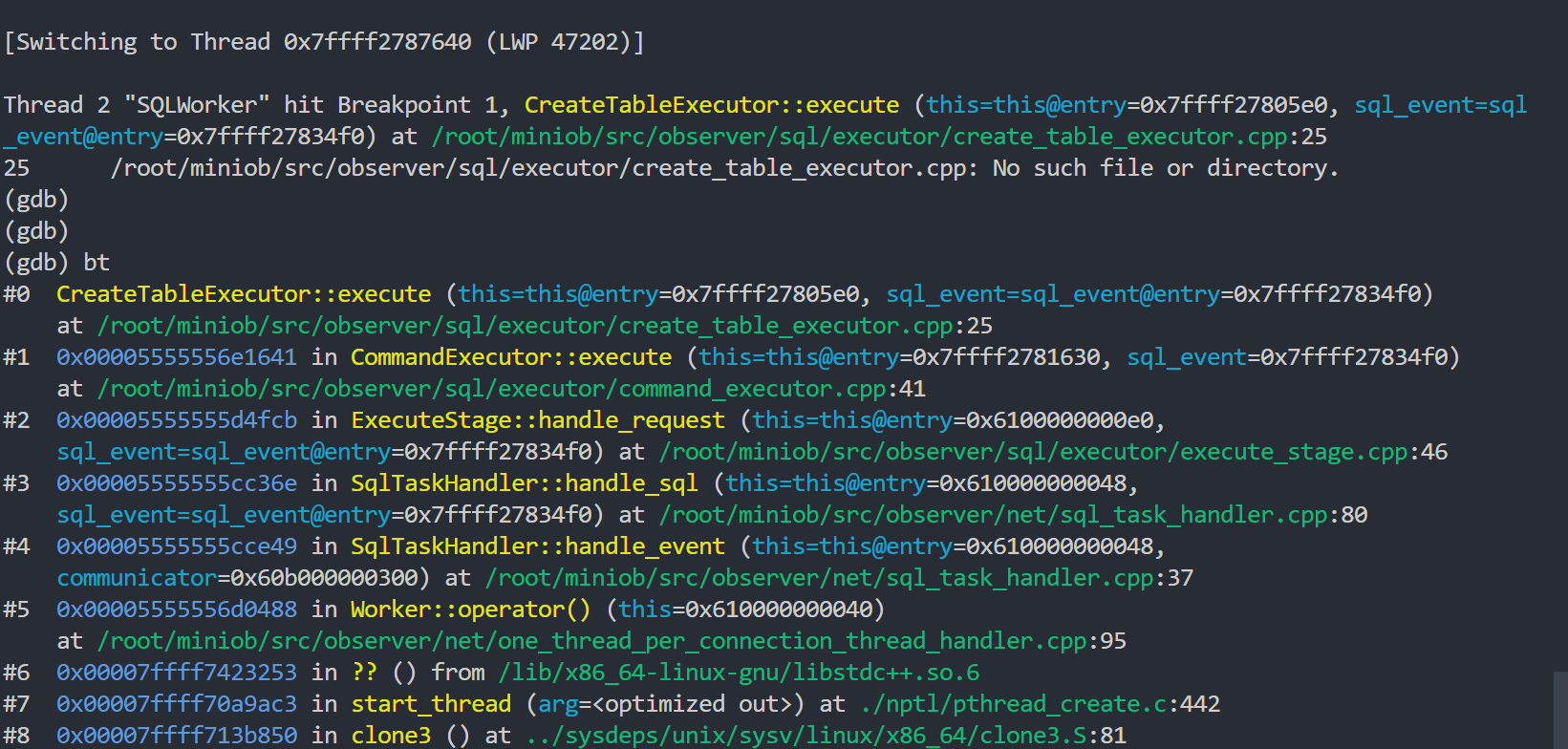
再开一个终端启动客户端



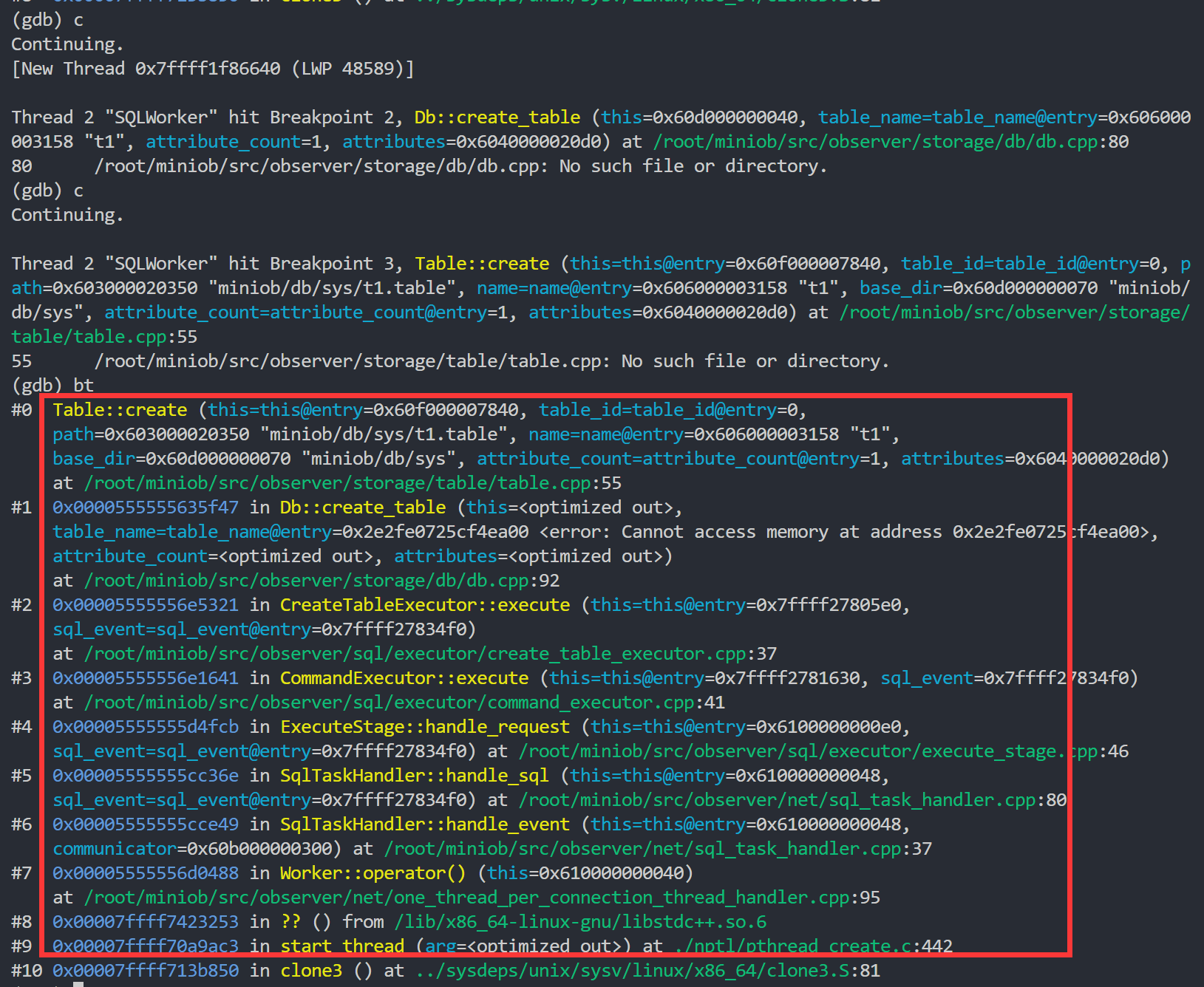
输入创建表sql



可以发现gdb 触发断点CreateTableExecutor::execute， 输入bt查看当前栈帧



输入c继续执行，直到hit下一个断点



这就是创建表整个流程的栈帧，逐步分析这些函数，进而了解创建表过程整个逻辑。

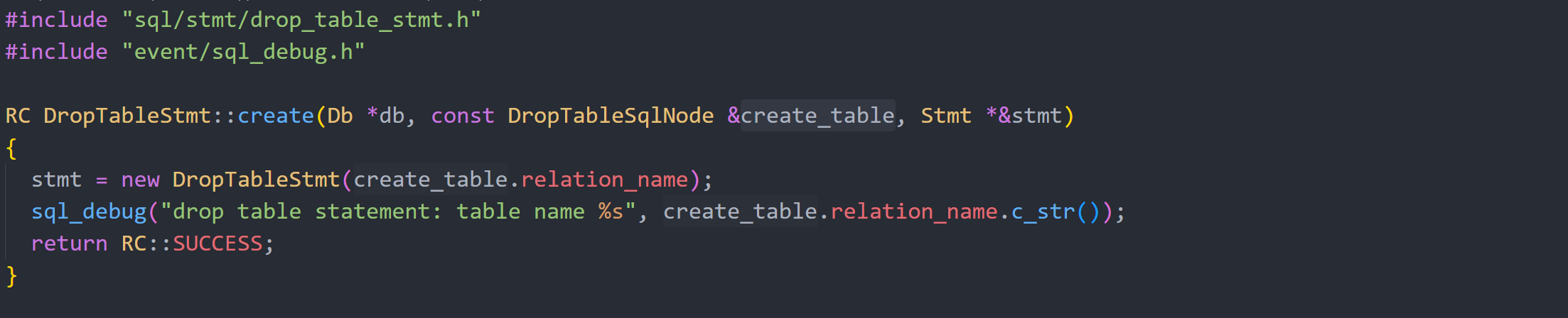
1. 设计Drop Table实现

根据创建表的流程，可知道整个sql的执行过程。

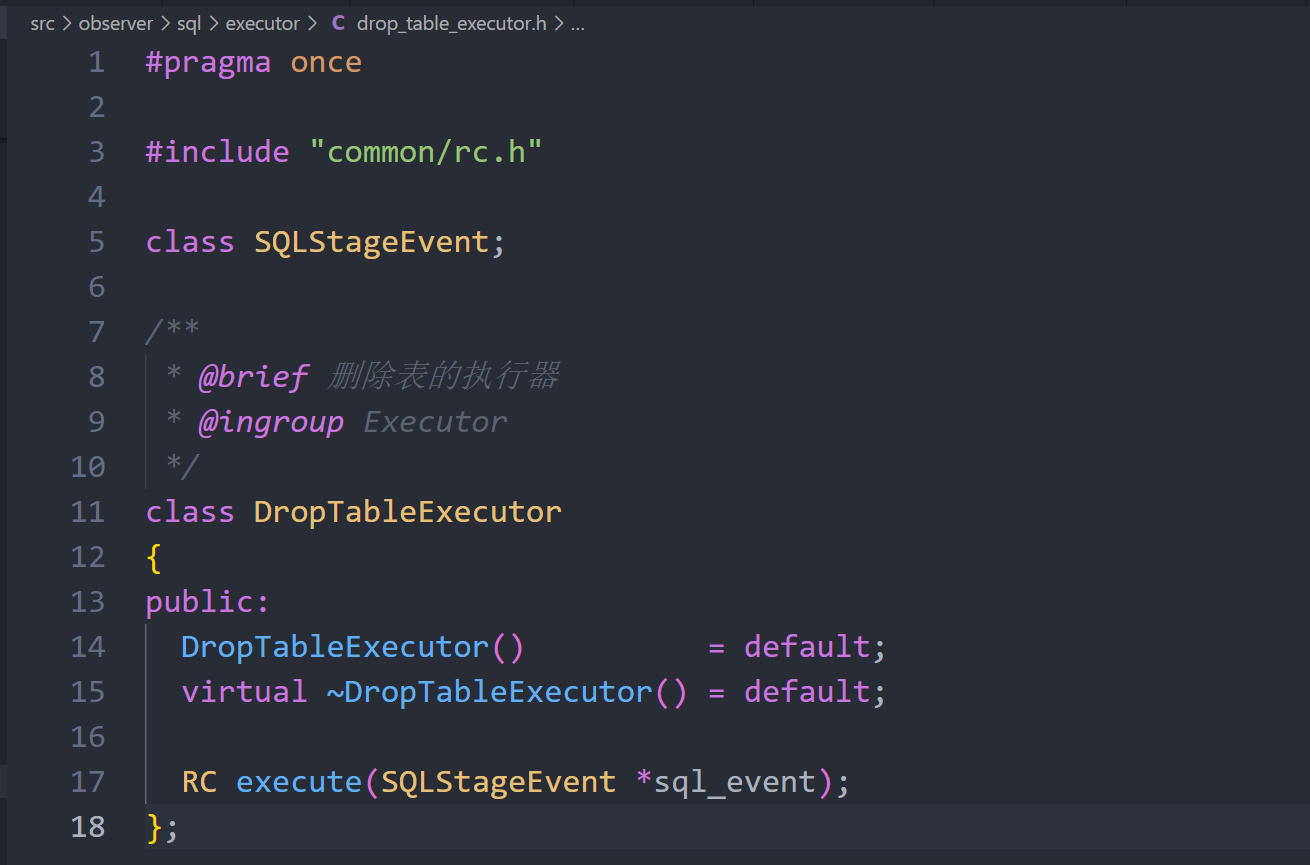
先根据create\_table\_executor执行器代码中的CreateTableStmt先添加drop\_table对应的stmt

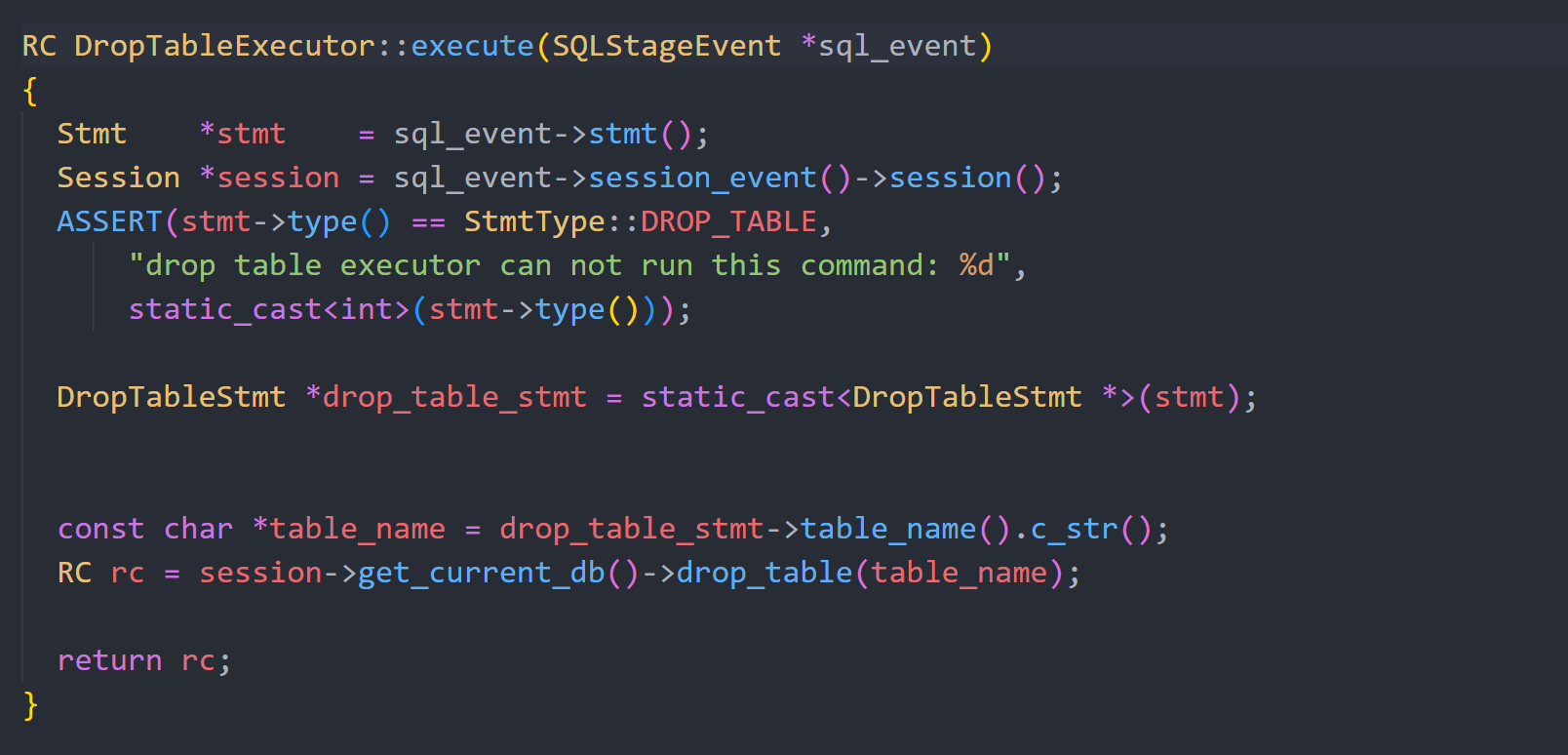
DropTableSqlNode描述一个drop table语句





再按照create\_table\_executor执行器按葫芦画瓢写出drop table的执行器。





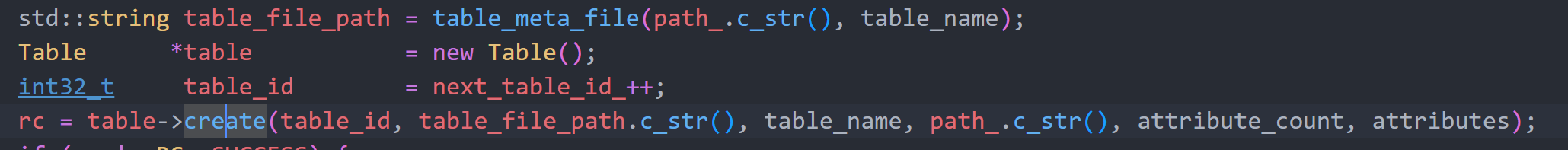
在Db类中加入drop table执行器所需要调用的函数drop\_table;

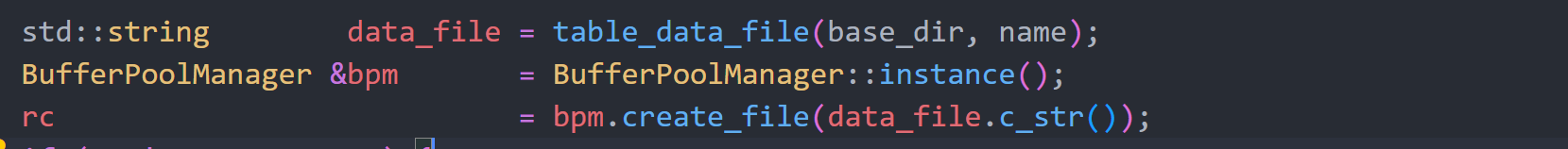




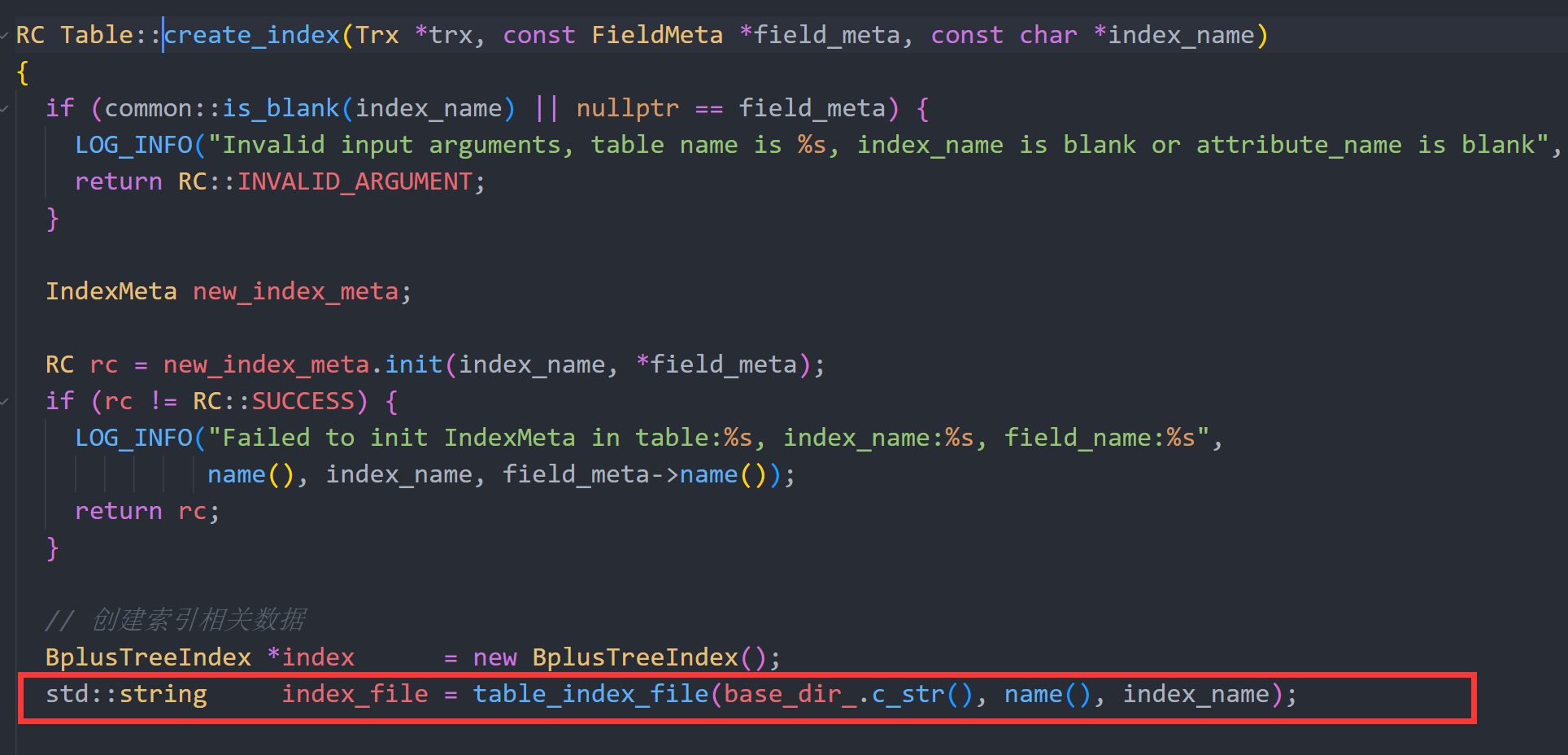
在Table类中实现drop函数，函数实现主要为删除其表对应的文件和这张表的索引文件。

根据创建表代码得知表文件所在路径，droptable则需要删除这些文件





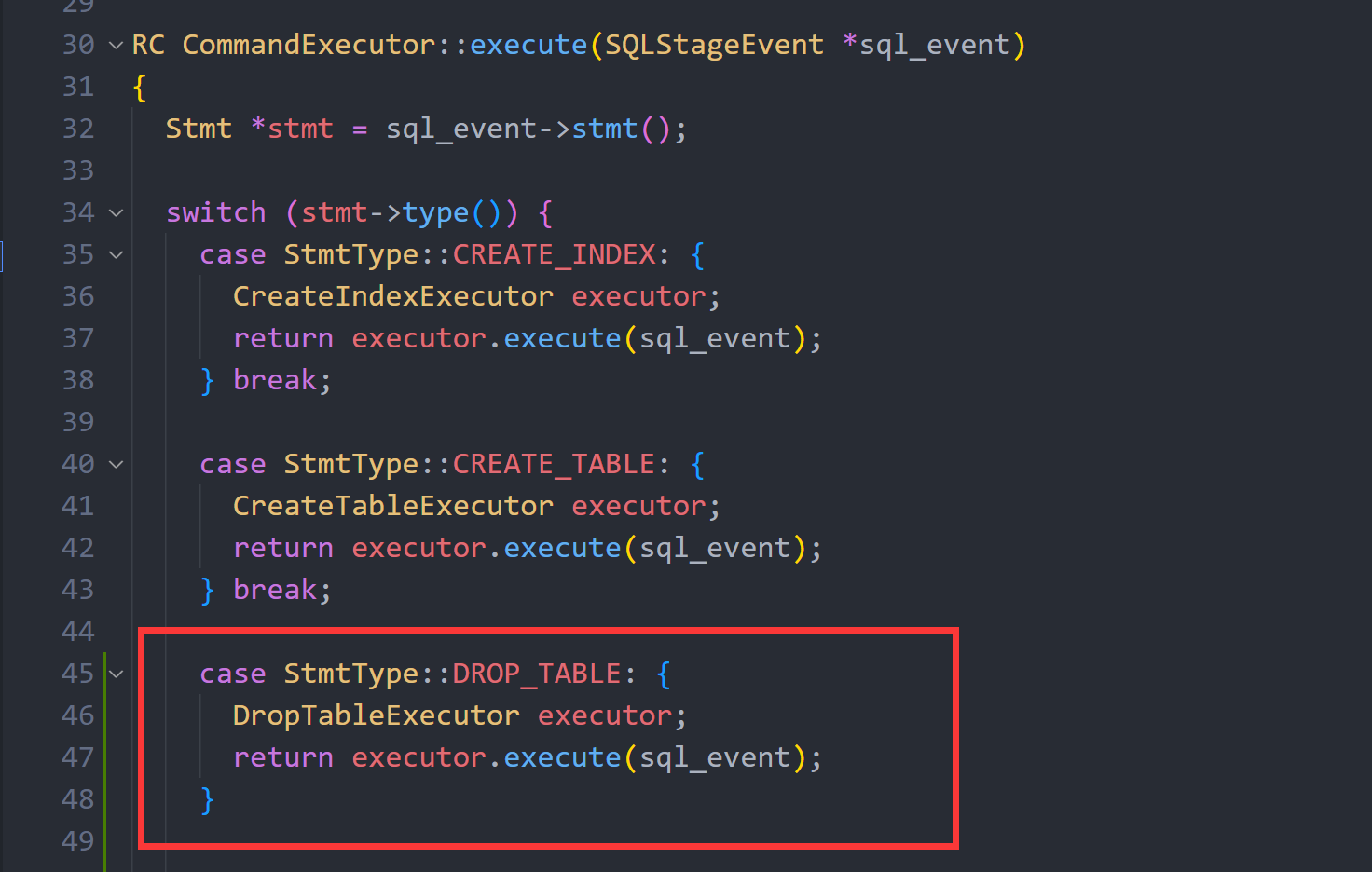
根据table::create\_index 的代码得知这张表的索引文件路径



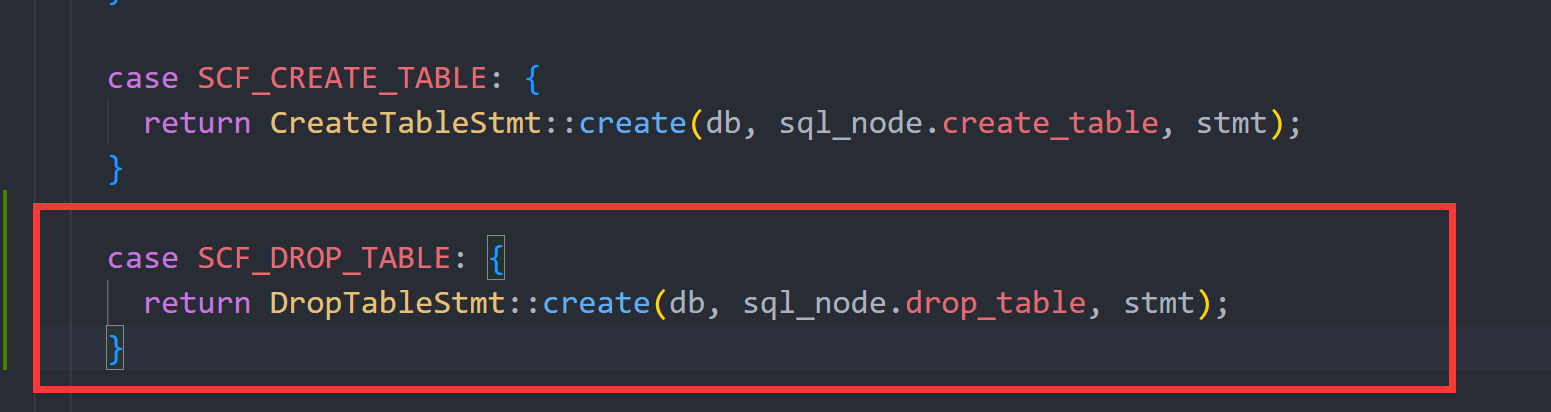
drop函数在文件系统中删除这些文件



在CommanExecutor::execute中加入Drop Table的对应的case



在stmt.cpp中加入创建删除表对应的case



最后编译 bash build.sh

启动observer

再启动client测试成功

